

CCCCCCCCCCCC	DDDDDDDDDDDD	UUU	UUU
CCCCCCCCCCCC	DDDDDDDDDDDD	UUU	UUU
CCCCCCCCCCCC	DDDDDDDDDDDD	UUU	UUU
CCC	DDD	UUU	UUU
CCC	DDD	UUU	UUU
CCC	DDD	UUU	UUU
CCC	DDD	UUU	UUU
CCC	DDD	UUU	UUU
CCC	DDD	UUU	UUU
CCC	DDD	UUU	UUU
CCC	DDD	UUU	UUU
CCC	DDD	UUU	UUU
CCC	DDD	UUU	UUU
CCC	DDD	UUU	UUU
CCC	DDD	UUU	UUU
CCC	DDD	UUU	UUU
CCC	DDD	UUU	UUU
CCC	DDD	UUU	UUU
CCC	DDD	UUU	UUU
CCC	DDD	UUU	UUU
CCCCCCCCCCCC	DDDDDDDDDDDD	UUUUUUUUUUUUUUUU	UUUUUUUUUUUUUUUU
CCCCCCCCCCCC	DDDDDDDDDDDD	UUUUUUUUUUUUUUUU	UUUUUUUUUUUUUUUU
CCCCCCCCCCCC	DDDDDDDDDDDD	UUUUUUUUUUUUUUUU	UUUUUUUUUUUUUUUU

```
PPPPPPPP      AAAAAA      RRRRRRRR      SSSSSSSS      EEEEEEEEEE      333333
PPPPPPPP      AAAAAA      RRRRRRRR      SSSSSSSS      EEEEEEEEEE      333333
PP      PP      AA      AA      RR      RR      SS      EEEEEEEEEE      33      33
PP      PP      AA      AA      RR      RR      SS      EEEEEEEEEE      33      33
PP      PP      AA      AA      RR      RR      SS      EEEEEEEEEE      33      33
PP      PP      AA      AA      RR      RR      SS      EEEEEEEEEE      33      33
PPPPPPPP      AA      AA      RRRRRRRR      SSSSSS      EEEEEEEEEE      33
PPPPPPPP      AA      AA      RRRRRRRR      SSSSSS      EEEEEEEEEE      33
PP      AAAAAAAAAA      RR      RR      SS      EEEEEEEEEE      33
PP      AAAAAAAAAA      RR      RR      SS      EEEEEEEEEE      33
PP      AA      AA      RR      RR      SS      EEEEEEEEEE      33
PP      AA      AA      RR      RR      SS      EEEEEEEEEE      33
PP      AA      AA      RR      RR      SSSSSSSS      EEEEEEEEEE      333333
PP      AA      AA      RR      RR      SSSSSSSS      EEEEEEEEEE      333333
                                     ....
                                     ....
                                     ....
                                     ....
```

```
LL      IIIIII      SSSSSSSS
LL      IIIIII      SSSSSSSS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SSSSSS
LL      II      SSSSSS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SS
LLLLLLLLLL      IIIIII      SSSSSSSS
LLLLLLLLLL      IIIIII      SSSSSSSS
```



```

1 0001 0 MODULE parse3 (IDENT='V04-000'
2 0002 0 ADDRESSING_MODE(EXTERNAL=GENERAL))
3 0003 1 = BEGIN
4 0004 1
5 0005 1 *****
6 0006 1 *
7 0007 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
8 0008 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
9 0009 1 * ALL RIGHTS RESERVED.
10 0010 1 *
11 0011 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
12 0012 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
13 0013 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
14 0014 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
15 0015 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
16 0016 1 * TRANSFERRED.
17 0017 1 *
18 0018 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
19 0019 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
20 0020 1 * CORPORATION.
21 0021 1 *
22 0022 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
23 0023 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
24 0024 1 *
25 0025 1 *
26 0026 1 *****
27 0027 1
28 0028 1 ++
29 0029 1 Facility: Command Definition Utility, CLD Parser Module 3
30 0030 1
31 0031 1 Abstract: This module is one of a few modules that implements the
32 0032 1 parser for CLD files. This parser translates the CLD source
33 0033 1 language into an intermediate representation composed of
34 0034 1 nodes linked in a directed graph.
35 0035 1
36 0036 1 Environment: Standard CDU environment.
37 0037 1
38 0038 1 Author: Paul C. Anagnostopoulos
39 0039 1 Creation: 6 December 1982
40 0040 1
41 0041 1 Modifications:
42 0042 1
43 0043 1 V04-002 PCG0002 Peter George 07-Dec-1983
44 0044 1 Allow NOT to precede expressions.
45 0045 1
46 0046 1 V04-001 PCG0001 Peter George 06-Dec-1983
47 0047 1 Add NEG operator.
48 0048 1
49 0049 1 --
50 0050 1
51 0051 1
52 0052 1 Library 'sys$library:lib';
53 0053 1 require 'cdureq';

```

55	0467	1	!	T A B L E O F C O N T E N T S
56	0468	1	!	-----
57	0469	1		
58	0470	1	forward	routine
59	0471	1		cdu\$bool_expr,
60	0472	1		cdu\$bool_term,
61	0473	1		cdu\$bool_factor,
62	0474	1		cdu\$path;
63	0475	1		
64	0476	1		
65	0477	1	!	E X T E R N A L R E F E R E N C E S
66	0478	1	!	-----
67	0479	1		
68	0480	1	external	routine
69	0481	1		cdu\$check_for_children,
70	0482	1		cdu\$create_node,
71	0483	1		cdu\$get_next_token,
72	0484	1		cdu\$lookup_child,
73	0485	1		cdu\$report_syntax_error,
74	0486	1		cdu\$token_must_be;
75	0487	1		
76	0488	1	external	
77	0489	1		cdu\$gl_token_class: long,
78	0490	1		cdu\$gq_token: descriptor;


```

80 0491 1  !++
81 0492 1  Description: This syntax routine recognizes the 'bool-expr' construct,
82 0493 1  which is the top-level boolean expression construct.
83 0494 1
84 0495 1  Parameters: None.
85 0496 1
86 0497 1  Returns: Top-level node representing a boolean expression.
87 0498 1
88 0499 1  Notes:
89 0500 1  --
90 0501 1
91 0502 1 GLOBAL ROUTINE cdu$bool_expr
92 0503 2 = BEGIN
93 0504 2
94 0505 2 local
95 0506 2     expr: ref node,
96 0507 2     term: ref node,
97 0508 2     last_term: ref node;
98 0509 2
99 0510 2
100 0511 2 ! A boolean expression must begin with a boolean term. Parse the term.
101 0512 2 ! If there is no operator following it, then return the term as
102 0513 2 ! the representation of the expression.
103 0514 2
104 0515 2 term = cdu$bool_term();
105 0516 2 if not token_is(tkn_k_symbol,'OR') then
106 0517 2     return .term;
107 0518 2
108 0519 2 ! Create an OR node to collect the terms.
109 0520 2
110 0521 2 expr = cdu$create_node(node_k_or);
111 0522 2
112 0523 2 ! Now go into a loop to recognize all remaining operators and terms.
113 0524 2
114 0525 2 loop (
115 0526 2
116 0527 2     ! Link the latest term onto the expression node.
117 0528 2     link_parent_to_child(expr,term,last_term);
118 0529 2
119 0530 2     ! If we don't have another operator, then we are done.
120 0531 2     if not token_is(tkn_k_symbol,'OR') then exitloop;
121 0532 2
122 0533 2     ! We must have another term.
123 0534 2
124 0535 2     cdu$get_next_token();
125 0536 2     term = cdu$bool_term();
126 0537 2
127 0538 2 );
128 0539 2
129 0540 2 return .expr;
130 0541 2
131 0542 2
132 0543 1 END;

```

INFO#250 L1:0529
Referenced LOCAL symbol LAST_TERM is probably not initialized

				.TITLE	PARSE3	
				.IDENT	\V04-000\	
				.PSECT	SPLITS,NOWRT,NOEXE,2	
52	4F	00000	P.AAA:	.ASCII	\OR\	:
52	4F	00002	P.AAB:	.ASCII	\OR\	:
				.EXTRN	CDUSCHECK FOR CHILDREN	
				.EXTRN	CDUSCREATE NODE	
				.EXTRN	CDUSGET NEXT TOKEN	
				.EXTRN	CDUSLOOKUP CHILD	
				.EXTRN	CDUSREPORT SYNTAX_ERROR	
				.EXTRN	CDUSTOKEN MUST BE	
				.EXTRN	CDUSGL_TOKEN_CLASS	
				.EXTRN	CDUSGQ_TOKEN	
				.PSECT	\$CODE\$,NOWRT,2	
			01FC 00000	.ENTRY	CDUS\$BOOL_EXPR, Save R2,R3,R4,R5,R6,R7,R8	: 0502
		58	00000000G 00 9E 00002	MOVAB	CDUSGL_TOKEN_CLASS, R8	:
		57	00000000G 00 9E 00009	MOVAB	CDUSGQ_TOKEN+4, R7	:
0000V		CF	00 FB 00010	CALLS	#0, CDUS\$BOOL_TERM	: 0515
		56	50 D0 00015	MOVL	R0, TERM	:
		0D	68 D1 00018	CMPL	CDUSGL_TOKEN_CLASS, #13	: 0516
			0E 12 0001B	BNEQ	1\$	:
02	00	50	67 D0 0001D	MOVL	CDUSGQ_TOKEN+4, R0	:
		60	A7 2D 00020	CMPC5	CDUSGQ_TOKEN, (R0), #0, #2, P.AAA	:
			CF 00026			:
			04 13 00029	BEQL	2\$	:
		50	56 D0 0002B 1\$:	MOVL	TERM, R0	: 0517
			04 0002E	RET		:
			29 DD 0002F 2\$:	PUSHL	#41	: 0521
	00000000G	00	01 FB 00031	CALLS	#1, CDUS\$CREATE_NODE	:
		55	50 D0 00038	MOVL	R0, EXPR	:
			08 A5 D5 0003B 3\$:	TSTL	8(EXPR)	: 0529
			06 12 0003E	BNEQ	4\$	:
	08	A5	56 D0 00040	MOVL	TERM, 8(EXPR)	:
			04 11 00044	BRB	5\$	:
	04	A4	56 D0 00046 4\$:	MOVL	TERM, 4(LAST_TERM)	:
		54	56 D0 0004A 5\$:	MOVL	TERM, LAST_TERM	:
		0D	68 D1 0004D	CMPL	CDUSGL_TOKEN_CLASS, #13	: 0533
			1F 12 00050	BNEQ	6\$	:
		50	67 D0 00052	MOVL	CDUSGQ_TOKEN+4, R0	:
02	00	60	A7 2D 00055	CMPC5	CDUSGQ_TOKEN, (R0), #0, #2, P.AAB	:
			CF 0005B			:
			11 12 0005E	BNEQ	6\$	:
	00000000G	00	00 FB 00060	CALLS	#0, CDUS\$GET_NEXT_TOKEN	: 0537
	0000V	CF	00 FB 00067	CALLS	#0, CDUS\$BOOC_TERM	: 0538
		56	50 D0 0006C	MOVL	R0, TERM	:
			CA 11 0006F	BRB	3\$	: 0521
		50	55 D0 00071 6\$:	MOVL	EXPR, R0	: 0541
			04 00074	RET		: 0543

; Routine Size: 117 bytes, Routine Base: \$CODE\$ + 0000


```

134 0544 1 !++
135 0545 1 ! Description: This syntax routine recognizes the 'bool-term' construct,
136 0546 1 ! which is the construct from which boolean expressions are
137 0547 1 ! built.
138 0548 1
139 0549 1 ! Parameters: None.
140 0550 1
141 0551 1 ! Returns: The top-level node representing the term.
142 0552 1
143 0553 1 ! Notes:
144 0554 1 ! --
145 0555 1
146 0556 1 GLOBAL ROUTINE cdu$bool_term
147 0557 2 = BEGIN
148 0558 2
149 0559 2 local
150 0560 2     term: ref node,
151 0561 2     factor: ref node,
152 0562 2     last_factor: ref node;
153 0563 2
154 0564 2
155 0565 2 ! A boolean term must begin with a boolean factor. Parse the factor.
156 0566 2 ! If there is no operator following it, then return the factor as
157 0567 2 ! the representation of the term.
158 0568 2
159 0569 2 factor = cdu$bool_factor();
160 0570 2 if not token_is(tkn_k_symbol,'AND') then
161 0571 2     return .factor;
162 0572 2
163 0573 2 ! Create an AND node to collect the factors.
164 0574 2
165 0575 2 term = cdu$create_node(node_k_and);
166 0576 2
167 0577 2 ! Now go into a loop to recognize all remaining operators and factors.
168 0578 2
169 0579 2 loop (
170 0580 2
171 0581 2     ! Link the latest factor onto the term node.
172 0582 2     link_parent_to_child(term,factor,last_factor);
173 0583 2
174 0584 2     ! If we don't have another operator, then we are done.
175 0585 2     if not token_is(tkn_k_symbol,'AND') then exitloop;
176 0586 2
177 0587 2     ! We must have another factor.
178 0588 2     cdu$get_next_token();
179 0589 2     factor = cdu$bool_factor();
180 0590 2
181 0591 2 );
182 0592 2
183 0593 2 return .term;
184 0594 2
185 0595 2
186 0596 2
187 0597 1 END;
```

INFO#250 L1:0583
Referenced LOCAL symbol LAST_FACTOR is probably not initialized


```

189 0598 1 !++
190 0599 1 ! Description: This syntax routine recognizes a 'bool-factor' construct,
191 0600 1 ! which is the construct from which boolean terms are built.
192 0601 1
193 0602 1 Parameters: None.
194 0603 1
195 0604 1 Returns: The top-level node representing the factor.
196 0605 1
197 0606 1 Notes:
198 0607 1 --
199 0608 1
200 0609 1 GLOBAL ROUTINE cdu$bool_factor
201 0610 2 = BEGIN
202 0611 2
203 0612 2 local
204 0613 2 factor: ref node,
205 0614 2 item: ref node,
206 0615 2 last_item: ref node;
207 0616 2
208 0617 2
209 0618 2 ! Determine which kind of factor we have.
210 0619 2
211 0620 2 if token_is(tkn_k_open_paren) then (
212 0621 2
213 0622 2 ! We have a subexpression. Just return its top-level node.
214 0623 2
215 0624 2 cdu$get_next_token();
216 0625 2 factor = cdu$bool_expr();
217 0626 2 cdu$token_must_be(tkn_k_close_paren);
218 0627 2 return .factor;
219 0628 2 );
220 0629 2
221 0630 2 if token_is(tkn_k_symbol,'ANY2') then (
222 0631 2
223 0632 2 ! We have an ANY2 operator, which really looks like a function.
224 0633 2 ! Create a node to represent the ANY2 function.
225 0634 2
226 0635 2 factor = cdu$create_node(node_k_any2);
227 0636 2 cdu$get_next_token();
228 0637 2
229 0638 2 ! We have a parenthesized list of paths.
230 0639 2 ! Eat the open parenthesis. Then sit in a loop which recognizes at
231 0640 2 ! least one path, along with any others separated by commas. Finally,
232 0641 2 ! eat the close parenthesis. All of the paths are chained as children
233 0642 2 ! of the ANY2 node.
234 0643 2
235 0644 2 cdu$token_must_be(tkn_k_open_paren);
236 0645 2 loop (
237 0646 2 item = cdu$path();
238 0647 2 link_parent_to_child(factor,item,last_item);
239 0648 2 if not token_is(tkn_k_comma) then exit loop;
240 0649 2 cdu$get_next_token();
241 0650 2 );
242 0651 2 cdu$token_must_be(tkn_k_close_paren);
243 0652 2 return .factor;
244 0653 2 );
245 0654 2

```

```

246 0655 3 if token_is(tkn_k_symbol,'NOT') then (
247 0656
248 0657 ! We have a NOT operator. Create a node to represent it. Recognize
249 0658 ! a boolean factor following the NOT and link it as the child.
250 0659
251 0660 factor = cdu$create_node(node_k_not);
252 0661 cdu$get_next_token();
253 0662 item = cdu$bool_factor();
254 0663 link_parent_to_child(factor,item,last_item);
255 0664 return .factor;
256 0665 );
257 0666
258 0667 if token_is(tkn_k_symbol,'NEG') then (
259 0668
260 0669 ! We have a NEG operator. Create a node to represent it. Recognize
261 0670 ! a path following the NEG and link it as the child.
262 0671
263 0672 factor = cdu$create_node(node_k_neg);
264 0673 cdu$get_next_token();
265 0674 item = cdu$path();
266 0675 link_parent_to_child(factor,item,last_item);
267 0676 return .factor;
268 0677 );
269 0678
270 0679 ! If it's none of the above, then it must be a path, which specifies a
271 0680 ! particular parameter, qualifier, or type keyword.
272 0681
273 0682 return cdu$path();
274 0683
275 0684 1 END;

```

INFO#250 L1:0647
Referenced LOCAL symbol LAST_ITEM is probably not initialized

.PSECT \$SPLITS,NOWRT,NOEXE,2

```

32 59 4E 41 0000A P.AAE: .ASCII \ANY2\
54 4F 4E 0000E P.AAF: .ASCII \NOT\
47 45 4E 00011 P.AAG: .ASCII \NEG\

```

.PSECT \$CODE\$,NOWRT,2

```

OFFC 0000Q
5B 00000000G 00 9E 00002 MOVAB CDUSTOKEN MUST BE, R11
5A 00000000G 00 9E 00009 MOVAB CDUSCREATE NODE, R10
59 00000000G 00 9E 00010 MOVAB CDUSGL_TOKEN_CLASS, R9
58 00000000G 00 9E 00017 MOVAB CDUSGET_NEXT_TOKEN, R8
57 00000000G 00 9E 0001E MOVAB CDUSGQ_TOKEN+4, R7
07 69 D1 00025 CMPL CDUSGL_TOKEN_CLASS, #7
0D 12 00028 BNEQ 1$
68 00 FB 0002A CALLS #0, CDUSGET_NEXT_TOKEN
FEE4 CF 00 FB 0002D CALLS #0, CDUSBOOC_EXPR
54 50 D0 00032 MOVL R0, FACTOR

```


04	00	0D	FC	47 11 00035	BRB	5\$	0626
		50	0000	69 D1 00037 1\$:	CMP	CDU\$GL_TOKEN_CLASS, #13	0630
		60		49 12 0003A	BNEQ	6\$	
				67 D0 0003C	MOVL	CDU\$GQ_TOKEN+4, R0	
				A7 2D 0003F	CMPC5	CDU\$GQ_TOKEN, (R0), #0, #4, P.AAE	
				CF 00045			
				3B 12 00048	BNEQ	6\$	
		6A		2D DD 0004A	PUSHL	#45	0635
		54		01 FB 0004C	CALLS	#1, CDU\$CREATE_NODE	
		68		50 D0 0004F	MOVL	R0, FACTOR	
				00 FB 00052	CALLS	#0, CDU\$GET_NEXT_TOKEN	0636
		6B		07 DD 00055	PUSHL	#7	0644
	0000V	CF		01 FB 00057	CALLS	#1, CDU\$TOKEN_MUST_BE	
		56		00 FB 0005A 2\$:	CALLS	#0, CDU\$PATH	0646
			08	50 D0 0005F	MOVL	R0, ITEM	
				A4 D5 00062	TSTL	8(FACTOR)	0647
				06 12 00065	BNEQ	3\$	
	08	A4		56 D0 00067	MOVL	ITEM, 8(FACTOR)	
				04 11 0006B	BRB	4\$	
	04	A5		56 D0 0006D 3\$:	MOVL	ITEM, 4(LAST_ITEM)	
		55		56 D0 00071 4\$:	MOVL	ITEM, LAST_ITEM	
		05		69 D1 00074	CMP	CDU\$GL_TOKEN_CLASS, #5	0648
		68		05 12 00077	BNEQ	5\$	
				00 FB 00079	CALLS	#0, CDU\$GET_NEXT_TOKEN	0649
				DC 11 0007C	BRB	2\$	0644
		6B		08 DD 0007E 5\$:	PUSHL	#8	0651
				01 FB 00080	CALLS	#1, CDU\$TOKEN_MUST_BE	
				5D 11 00083	BRB	11\$	0652
		0D		69 D1 00085 6\$:	CMP	CDU\$GL_TOKEN_CLASS, #13	0655
				2C 12 00088	BNEQ	8\$	
		50		67 D0 0008A	MOVL	CDU\$GQ_TOKEN+4, R0	
03	00	60	FC	A7 2D 0008D	CMPC5	CDU\$GQ_TOKEN, (R0), #0, #3, P.AAF	
			0000	CF 00093			
				1E 12 00096	BNEQ	8\$	
				2C DD 00098	PUSHL	#44	0660
		6A		01 FB 0009A	CALLS	#1, CDU\$CREATE_NODE	
		54		50 D0 0009D	MOVL	R0, FACTOR	
		68		00 FB 000A0	CALLS	#0, CDU\$GET_NEXT_TOKEN	0661
	FF58	CF		00 FB 000A3	CALLS	#0, CDU\$BOOE_FACTOR	0662
		56		50 D0 000A8 7\$:	MOVL	R0, ITEM	
			08	A4 D5 000AB	TSTL	8(FACTOR)	0663
				2B 12 000AE	BNEQ	9\$	
	08	A4		56 D0 000B0	MOVL	ITEM, 8(FACTOR)	
				29 11 000B4	BRB	10\$	
		0D		69 D1 000B6 8\$:	CMP	CDU\$GL_TOKEN_CLASS, #13	0667
				2B 12 000B9	BNEQ	12\$	
		50		67 D0 000BB	MOVL	CDU\$GQ_TOKEN+4, R0	
03	00	60	FC	A7 2D 000BE	CMPC5	CDU\$GQ_TOKEN, (R0), #0, #3, P.AAG	
			0000	CF 000C4			
				1D 12 000C7	BNEQ	12\$	
				35 DD 000C9	PUSHL	#53	0672
		6A		01 FB 000CB	CALLS	#1, CDU\$CREATE_NODE	
		54		50 D0 000CE	MOVL	R0, FACTOR	
		68		00 FB 000D1	CALLS	#0, CDU\$GET_NEXT_TOKEN	0673
	0000V	CF		00 FB 000D4	CALLS	#0, CDU\$PATH	0674
				CD 11 000D9	BRB	7\$	
	04	A5		56 D0 000DB 9\$:	MOVL	ITEM, 4(LAST_ITEM)	0675

PARSE3
V04-000

C 13
15-Sep-1984 23:49:40 VAX-11 Bliss-32 V4.0-742 Page 10
14-Sep-1984 11:58:27 DISK\$VMSMASTER:[CDU.SRC]PARSE3.B32;1 (5)

55	56	DO	000DF	10\$:	MOVL	ITEM, LAST_ITEM	:	
50	54	DO	000E2	11\$:	MOVL	FACTOR, RO	:	0676
		04	000E5		RET		:	
0000V CF	00	FB	000E6	12\$:	CALLS	#0, CDUSPATH	:	0682
		04	000EB		RET		:	0684

; Routine Size: 236 bytes, Routine Base: \$CODE\$ + 00EA


```

: 277 0685 1 !++
: 278 0686 1 ! Description: This syntax routine recognizes the 'path' construct, which
: 279 0687 1 ! is used to specify an entity path. An entity path has the
: 280 0688 1 ! following format:
: 281 0689 1 !
: 282 0690 1 !     [< definition >] entity.entity...
: 283 0691 1 !
: 284 0692 1 ! The optional definition specifies a verb or syntax change
: 285 0693 1 ! definition, and the entities specify a hierarchical path
: 286 0694 1 ! to a parameter, qualifier, or type keyword.
: 287 0695 1 !
: 288 0696 1 ! Parameters: None.
: 289 0697 1 !
: 290 0698 1 ! Returns: The node representing the path.
: 291 0699 1 !
: 292 0700 1 ! Notes:
: 293 0701 1 ! --
: 294 0702 1
: 295 0703 1 GLOBAL ROUTINE cdu$path
: 296 0704 2 = BEGIN
: 297 0705 2
: 298 0706 2 local
: 299 0707 2     path: ref node,
: 300 0708 2     item: ref node,
: 301 0709 2     last_item: ref node;
: 302 0710 2
: 303 0711 2
: 304 0712 2 ! Create a node to represent the path.
: 305 0713 2
: 306 0714 2 path = cdu$create_node(node_k_path);
: 307 0715 2
: 308 0716 2 ! See if a definition name is specified in angle brackets.
: 309 0717 2
: 310 0718 2 if token_is(tkn_k_open_angle) then (
: 311 0719 2
: 312 0720 2     ! The next token must be a symbol specifying the definition.
: 313 0721 2     ! Create a node for it and link it to the top-level path node.
: 314 0722 2
: 315 0723 2     cdu$get_next_token();
: 316 0724 2     item = cdu$create_node(node_k_path_definition, .cdu$gq_token[len], .cdu$gq_token[ptr]);
: 317 0725 2     link_parent_to_child(path, item, last_item);
: 318 0726 2     cdu$token_must_be(tkn_k_symbol);
: 319 0727 2
: 320 0728 2     ! Now there must be a close angle bracket.
: 321 0729 2
: 322 0730 2     cdu$token_must_be(tkn_k_close_angle);
: 323 0731 2 );
: 324 0732 2
: 325 0733 2 ! Now we have a sequence of one or more symbols specifying the entity path.
: 326 0734 2 ! Create a node for each one and link them to the top-level path node.
: 327 0735 2
: 328 0736 2 loop (
: 329 0737 2     item = cdu$create_node(node_k_path_entity, .cdu$gq_token[len], .cdu$gq_token[ptr]);
: 330 0738 2     link_parent_to_child(path, item, last_item);
: 331 0739 2     cdu$token_must_be(tkn_k_symbol);
: 332 0740 2     if not token_is(tkn_k_dot) then exitloop;
: 333 0741 2     cdu$get_next_token();

```

PARSE3
V04-000

E 13
15-Sep-1984 23:49:40
14-Sep-1984 11:58:27

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[CDU.SRC]PARSE3.B32;1 Page 12
(6)

```
: 334      0742 2 );  
: 335      0743 2  
: 336      0744 2 return .path;  
: 337      0745 2  
: 338      0746 1 END;  
: INFO#250      L1:0725  
: Referenced LOCAL symbol LAST_ITEM is probably not initialized
```

			03FC	00000		.ENTRY	CDU\$PATH, Save R2,R3,R4,R5,R6,R7,R8,R9		0703
59	000000000G	00	9E	00002		MOVAB	CDU\$GET_NEXT_TOKEN, R9		
58	000000000G	00	9E	00009		MOVAB	CDU\$GL_TOKEN_CLASS, R8		
57	000000000G	00	9E	00010		MOVAB	CDU\$CREATE_NODE, R7		
56	000000000G	00	9E	00017		MOVAB	CDU\$TOKEN_MUST_BE, R6		
55	000000000G	00	9E	0001E		MOVAB	CDU\$GQ_TOKEN+4, R5		
		2E	DD	00025		PUSHL	#46		0714
67		01	FB	00027		CALLS	#1, CDU\$CREATE_NODE		
52		50	DD	0002A		MOVL	R0, PATH		
0E		68	D1	0002D		CMPL	CDU\$GL_TOKEN_CLASS, #14		0718
		2D	12	00030		BNEQ	3\$		
69		00	FB	00032		CALLS	#0, CDU\$GET_NEXT_TOKEN		0723
		65	DD	00035		PUSHL	CDU\$GQ_TOKEN+4		0724
7E	FC	A5	3C	00037		MOVZWL	CDU\$GQ_TOKEN, -(SP)		
		30	DD	0003B		PUSHL	#48		
67		03	FB	0003D		CALLS	#3, CDU\$CREATE_NODE		
54		50	DD	00040		MOVL	R0, ITEM		
	08	A2	D5	00043		TSTL	8(PATH)		0725
		06	12	00046		BNEQ	1\$		
08	A2	54	DD	00048		MOVL	ITEM, 8(PATH)		
		04	11	0004C		BRB	2\$		
04	A3	54	DD	0004E	1\$:	MOVL	ITEM, 4(LAST_ITEM)		
53		54	DD	00052	2\$:	MOVL	ITEM, LAST_ITEM		
		0D	DD	00055		PUSHL	#13		0726
66		01	FB	00057		CALLS	#1, CDU\$TOKEN_MUST_BE		
		0F	DD	0005A		PUSHL	#15		0730
66		01	FB	0005C		CALLS	#1, CDU\$TOKEN_MUST_BE		
		65	DD	0005F	3\$:	PUSHL	CDU\$GQ_TOKEN+4		0737
7E	FC	A5	3C	00061		MOVZWL	CDU\$GQ_TOKEN, -(SP)		
		31	DD	00065		PUSHL	#49		
67		03	FB	00067		CALLS	#3, CDU\$CREATE_NODE		
54		50	DD	0006A		MOVL	R0, ITEM		
	08	A2	D5	0006D		TSTL	8(PATH)		0738
		06	12	00070		BNEQ	4\$		
08	A2	54	DD	00072		MOVL	ITEM, 8(PATH)		
		04	11	00076		BRB	5\$		
04	A3	54	DD	00078	4\$:	MOVL	ITEM, 4(LAST_ITEM)		
53		54	DD	0007C	5\$:	MOVL	ITEM, LAST_ITEM		
		0D	DD	0007F		PUSHL	#13		0739
66		01	FB	00081		CALLS	#1, CDU\$TOKEN_MUST_BE		
09		68	D1	00084		CMPL	CDU\$GL_TOKEN_CLASS, #9		0740
		05	12	00087		BNEQ	6\$		
69		00	FB	00089		CALLS	#0, CDU\$GET_NEXT_TOKEN		0741
		D1	11	0008C		BRB	3\$		0731
50		52	DD	0008E	6\$:	MOVL	PATH, R0		0744

PARSE3
V04-000

F 13
15-Sep-1984 23:49:40
14-Sep-1984 11:58:27

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[CDU.SRC]PARSE3.B32;1 Page 13
(6)

04 00091 RET

: 0746

: Routine Size: 146 bytes, Routine Base: \$CODE\$ + 01D6

: 339 0747 1 END
: 340 0748 0 ELUDOM

PSECT SUMMARY

Name	Bytes	Attributes
\$PLIT\$	20 NOVEC,NOWRT, RD	,NOEXE,NOSHR, LCL, REL, CON,NOPIC,ALIGN(2)
\$CODE\$	616 NOVEC,NOWRT, RD	, EXE,NOSHR, LCL, REL, CON,NOPIC,ALIGN(2)

Library Statistics

File	----- Total	Symbols Loaded	----- Percent	Pages Mapped	Processing Time
_\$255\$DUA28:[SYSLIB]LIB.L32;1	18619	4	0	1000	00:01.8

: Information: 4
: Warnings: 0
: Errors: 0

COMMAND QUALIFIERS

: BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LIS\$:PARSE3/OBJ=OBJ\$:PARSE3 MSRC\$:PARSE3/UPDATE=(ENH\$:PARSE3)

: Size: 616 code + 20 data bytes
: Run Time: 00:14.5
: Elapsed Time: 00:33.0
: Lines/CPU Min: 3095
: Lexemes/CPU-Min: 18182
: Memory Used: 125 pages
: Compilation Complete

0044 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY